ARMY RESEARCH LABORATORY

The U.S. Army Research Laboratory Dynamic Terrain Server

by Mark A. Thomas

ARL-TR-2962 April 2003

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2962 **April 2003**

The U.S. Army Research Laboratory Dynamic Terrain Server

Mark A. Thomas
Computational and Information Sciences Directorate, ARL

INTENTIONALLY LEFT BLANK.

Contents

Acknowledgments

The author would like to acknowledge the participation of the following individuals: Ms. Pat Jones, U.S. Army Research Laboratory (ARL), for managerial support; Mr. Andrew Neiderer and Mr. Charles Hansen, ARL, for coding and algorithm development; Mr. James Grosse and Mr. Brian Comer, Program Executive Office for Simulation, Training, and Instrumentation, for their support in funding the integration of the Dynamic Terrain Server (DTServer) with the Soldier Visualization System; and Dr. Bruce Knerr and Dr. Steve Goldberg, U.S. Army Research Institute for the Behavioral Sciences, for funding integration of the DTServer with the After-Action Review system developed by the Institute for Simulation and Technology.

1. Introduction

The U.S. Army Research Laboratory (ARL) is conducting research in methods to compute and distribute dynamic terrain information in real-time distributed simulation. The purpose is to populate virtual environments with real-world dynamic terrain and objects such as rubble and debris, dings, and breaches of structures. Dismounted infantry simulation requires these obstacles and capabilities for room-clearing operations, urban terrain warfare, and situational awareness.

The ARL Dynamic Terrain Server (DTServer) provides this capability. The DTServer computes effects on structures from munitions detonations and collisions. The server transmits the results to client simulations on the network and updates the status of rubble entities. The result of using the DTServer is a unified terrain database across all simulators ensuring fair-fight, realism, and training effectiveness. This report will describe the DTServer capabilities, architecture, and run-time considerations.

2. Capabilities

The DTServer computes effects on structures using table look-up and empirical formulas. The DTServer contains a munitions effects table which has data for relevant munitions such as the 9-mm bullet, 50-cal. munition, AT8 munition, and artillery. These munitions all create some effect on structures. A 9-mm impact on a concrete structure will leave a mark, or ding, on a concrete wall. This mark is not significant for structural mechanics but is significant to dismounted combatants for situational awareness and survivability. An AT8 blast, however, will completely breach a wall, affording access for room-clearing operations. The DTServer computes these effects and distributes them to compliant simulations across the network.

The DTServer transmits dynamic changes using distributed interactive simulation (DIS) protocol data units (PDU). The DIS Set Data PDU is used to transmit data to client simulations for breaches and dings. The Set Data PDUs sent by the DTServer are the Breach Subtract, Breach Vertices, and Ding messages.

Rubble is treated as an entity. This is a compatibility issue with the Dismounted Infantry Semi-Automated Forces (DISAF) program. The DISAF program requires geometry to be continuous. The breaching function cannot guarantee continuity with the rubble field computed. Therefore, each piece of rubble is transmitted to the simulators as an entity. The Entity State PDU transmits rubble state to client applications. The DTServer computes the rubble field and transmits a PDU

for each piece of rubble. The DTServer then sends a heartbeat for each piece of rubble to keep it active in the simulation.

The DTServer can log and replay events. Logged events can be replayed to provide an after-action capability or to predamage a database using data from a previous run.

The DTServer includes a graphical user interface (GUI) to view the database. The DTServer display is shown in Figure 1.

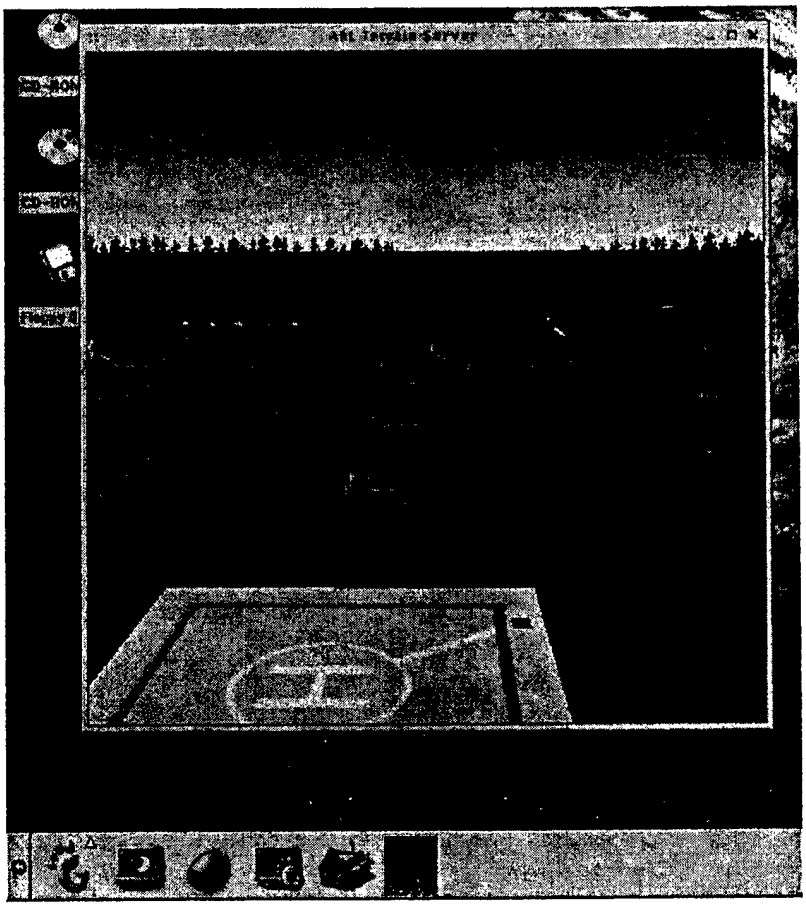

Figure 1. DTServer display showing Fort Polk database.

3. Architecture

The DTServer is comprised of a network interface unit (NIU), a munitions effects module (MEM), and a GUI.

3.1 Network Interface Unit

The NIU handles DIS communications for the DTServer. The DTServer reads the DIS Detonation PDU network packet. The Detonation PDU contains the information required for the munitions effects module. This information includes the point of detonation, the munition description, and the velocity vector. When a Detonation PDU is received by the NIU, it is sent to the munitions effects module.

The NIU requires two values at program initialization, the DIS port and exercise identifier. These values are entered by the user at program start-up.

3.2 Munitions Effects Module

The MEM computes the effect of the detonation on the impacted object. The MEM contains a database of munitions properties, terrain objects (buildings, bridges, other man-made objects), and ground terrain. The MEM parses the munition type, velocity vector, and detonation location from the Detonation PDU. If the detonation location does not impact a terrain object, the MEM does nothing. If a terrain object is impacted, the effect is computed. The computation is a simple mathematical relation between hole size and the mass of trinitrotoluene (TNT) in the projectile: $d = m_p 1/3$, where d = diameter and m_p = mass of TNT.

This relation was derived from a need to compute holes of different sizes from different munitions. The formula can be replaced by physics-based models which factor in projectile impact speed, direction, and wall properties. The effects computation returns two values—a breach flag and a size. If the breach flag is true, the size is the radius of the breach. If the breach flag is false, the size is the size of the ding.

If a breach occurs, the geometry engine is called to compute the new geometry. The new geometry consists of the new polygons which make up the new wall. When this computation is completed, the NIU is called to transmit the Set Data PDU Breach Subtract PDU and the set data polygon information. In addition, Entity State PDUs are transmitted for the resulting rubble.

If the breach flag is false, a ding is the result. The size is the radius of the ding. A Set Data PDU ding packet is formatted and sent to clients.

3.3 Set Data PDU Breach Subtract PDU

The Set Data PDU Breach Subtract PDU includes the necessary information for clients to initiate a breach. The PDU data packet includes breach location in round world coordinates, the surface normal in local coordinates, the name of the texture of the impacted polygon, and the number of polygons to follow. The following code fragment shows the construction of the Breach Subtract Set Data PDU:

```
typedef struct FixedDatumRecord{
  Unint32 id;
  Unint32 value;
}

typedef struct BreachRecord{
  Unint 32 id;
  Unint32 length;
  Float64 value[8];
}

typedef struct VertexRecord{
  Unint32 id;
  Unint32 length;
  Float64 value[5];
}

typedef struct DingRecord{
  Unint32 id;
  Unint32 length;
  Float64 value[5];
}

BreachRecord *v = (BreachRecord *)setdatapdu->variableDatum;
FixedDatumRecord *f = (FixedDatumRecord *)setdatapdu->fixedDatum;
```

```
setdatapdu->originating_entity_id.entity_id = EntityID;
setdatapdu->receiving_entity_id.address.site = 65535;
setdatapdu->receiving_entity_id.address.application = 65535;
setdatapdu->receiving_entity_id.entity_id = 65535;

setdatapdu->request_id = breach_number;
setdatapdu->number_of_fixed_datum_records = 34;
setdatapdu->number_of_variable_datum_records = 1;

f[0].id = BREACH_SUBTRACT_INFORMATION;
f[0].value = number_of_polygons;

f[1].id = BREACH_TEXTURE_INFORMATION;
f[1].value = strlen( texture);

for( int I = 0; I < 32; I++ ){
  f[I+2].id = BREACH_TEXTURE_INFORMATION + I + 1;
  f[I+2].value = texture[I];
}

v->id = BREACH_SUBTRACT_INFORMATION;
v->length = 512;
v->value[0] = world X location of breach;
v->value[1] = world Y location of breach;
v->value[2] = world Z location of breach;
v->value[3] = surface normal X;
```

v->value[4] = surface normal Y;

v->value[5] = surface normal Z;

v->value[6] = major_axis_length in meters;

v->value[7] = minor_axis_length in meters;

3.4 Breach Polygon Number Set Data PDU

The Breach Polygon Number Set Data PDU contains the polygon information. Clients will store the polygon data in the arrays allocated after arrival of the Breach Subtract Set Data PDU. The Breach Polygon Number Set Data PDU contains the number of the polygon from zero to N, the round world coordinates of the vertex, and the texture coordinates. The following code fragment shows the construction of the data packet:

FixedDatumRecord *f = (FixedDatumRecord *)setdatapdu->fixedDatum;

VertexRecord *v = (VertexRecord *)setdatapdu->variableDatum;

setdatapdu->originating_entity_id.entity_id = EntityID;

setdatapdu->receiving_entity_id.address.site = 65535;

setdatapdu->receiving_entity_id.address.application = 65535;

setdatapdu->receiving_entity_id.entity_id = 65535;

setdatapdu->request_id = Breach_ID;

setdatapdu->number_of_fixed_datum_records = 1;

f->id = BREACH_POLYGON_NUMBER;

f->value = Polygon Number [0 – N]

setdatapdu->number_of_variable_datum_records = 3;

v[I]->id = BREACH_POLYGON_INFORMATION;

v[I]->length = 320;

v[I]->value[0] = world X location;

v[I]->value[1] = world Y location;

v[I]->value[2] = world Z location;

v[I]->value[3] = texture u coordinate;

v[I]->value[4] = texture v coordinate;

The information in the PDU is independent triangles. Each triangle is described by its coordinates (vertices) and texture values. Using this data, the receiving simulator culls out the impacted wall from the scene and replaces it with the triangles received from the network. The receiving simulator can extract color information for the new triangles from the impacted triangle. In the future, color information should be included in the Set Data PDU, with a corresponding decrease in network response.

3.5 Ding Information Set Data PDU

The ding data packet contains the necessary data to display a ding resulting from a small-arms impact on a terrain object. The following code fragment shows the Ding Set Data PDU:

FixedDatumRecord *f = (FixedDatumRecord *)setdatapdu->fixedDatum;

DingRecord *d = (DingRecord *)setdatapdu->variableDatum;

setdatapdu->originating_entity_id.entity_id = EntityID;

setdatapdu->receiving_entity_id.address.site = 65535;

setdatapdu->receiving_entity_id.address.application = 65535;

setdatapdu->receiving_entity_id.entity_id = 65535;

setdatapdu->request_id = DING_ID;

setdatapdu->number_of_fixed_datum_records = 1;

setdatapdu->number_of_variable_datum_records = 1;

d->id = DING_ID;

d->length = 320;

d->value[0] = world X location;

d->value[1] = world Y location;

d->value[2] = world Z location;

d->value[3] = angle in X/Y plane;

d->value[4] = size in meters;

4. DTServer GUI

The DTServer GUI provides a visual check on the state of the database. Breaches and rubble are displayed. The GUI is a three-dimensional display of the terrain, and the viewer can fly around the terrain using keyboard commands. The GUI may be turned off to increase system response on computers with slow graphics hardware. The GUI also allows the user to fire munitions at walls to create damage in a stand-alone mode. The user can select the weapon to be fired, aim at a wall, and fire the weapon. The effect is calculated, and the appropriate Set Data PDU is transmitted. This capability is useful when inserting new munitions into the database or testing client program compatibility.

The arrow keys provide steering and speed control. The up and down arrows speed up and slow down forward and reverse motion. The left and right arrow keys turn the view left and right. The F11 and F12 keys raise and lower the eyepoint. The H key turns the GUI drawing off. This will greatly improve DTServer processing speed on slower machines. The L key will turn on logging. The logger will store all received detonation events that result in a breach or ding. The events are stored in a file, DTlog.dat in the current working directory, and can be replayed with the R command. This allows the DTServer to store simulation runs for playback later or to predamage a database using data from an earlier run.

The space bar fires the current munition. The munition is fired at the center of the screen. The user selects the weapon with the S key.

5. Runtime Considerations

The DTServer requires three databases in order to run—the terrain database, the object database, and the munitions database. The following sections describe these databases and give an example of a run script for the DTServer.

5.1 Terrain Database

The terrain database is used for ground clamping rubble. The terrain database is loaded using the following –t command line option: dtserver –t ftpolk_terrain.flt. Multiple terrain databases may be loaded, each with the –t flag.

5.2 Object Database

The DTServer can function on an entire database (with the –t runtime option) or on specific objects within a database. The presence of the file scene.cfg will determine specific objects or whether the entire database is processed.

The scene.cfg file is a user-created datafile of object names. The object names may be file names or display list names. For example, buildingL.flt will load in the OpenFlight file named *buildingL.flt* and will add it to the list of processed objects. In addition, *node3* will search the database for a node named *node3* and add it to the list of processed objects.

The DTServer creates a file called nodenames.dat when executed. A node is a graphics object in the scene graph of the graphical database. A node can have a name; such named nodes are of interest. This file lists the names of all named nodes in a database. The user then selects the objects to be processed from this list and adds them to the scene.cfg file. This is a trial and error process and is prone to error; hence, caution is advised.

The scene.cfg file is of the following format:

staticNodes

{

Xbld1G

Xbld2C

Xbld1G

Xbld4H

Xbld5F

Xbld6EE

Xbld6EE

Xbld10H

Q1room

Xbld11HH

Xbld12HH

Xbld13B

Xbld14H

Xbld15H

Xbld16H

Xbld17HH

Xbld18H

Xwatertower

Xbld20H

Xbld21AA

Xbld22B

Xbld23A

Xbld6EE

Xbld25E

Xbld26D

Xbld6EE

Xbld6EE

Xhbuilding

}

terrainNodes

{

tGround

}

The staticNodes keyword lists the nodes which will be processed for detonations. Using the UNIX* system grep command, the node names are extracted from the nodenames.dat file created by the DTServer. The grep pfGroup command will print out all named groups in the scene graph. These nodes will be the only nodes which will be processed for breaches. The terrainNodes keyword lists nodes which will be processed as terrain. Terrain is not breached but can show dings for ground impacts. This is useful when showing history of ground impacts and residue of smoke munitions.

In addition, the scene.cfg file may contain file names. The corresponding keywords would be staticFiles and terrainFiles.

5.3 Munitions Database

For the munitions effects models, the munitions file contains information required such as the name, DIS enumeration, the equivalent weight of TNT, burst radius, mass, warhead diameter, fuze, and muzzle velocity. The file format is as follows:

Start{

Name: Redeye

Kind: 3

Domain: 1

Country: 225

Category: 1

SubCategory: 1

EqWtTNT: 5.00

BurstRadius: 10.00

SmokeType: 1

---

* Unix is a registered trademark of The Open Group.

Mass: 0.50

Diameter: 0.25

Warhead: 0.00

Fuze: 0.00

Muzzle Velocity: 0.00

}

.

.

.

The current values in the munitions file are derived from open-source literature.

5.4 Run Script

The DTServer is best run from a shell script. The following shell script correctly sets up all environmental variables and executes the DTServer:

```
#!/bin/csh

setenv DISIMDEV_ROOT /home/disim
setenv PFPATH $DISIMDEV_ROOT/texture:$DISIMDEV_ROOT/FTPOLK/textures
setenv LD_LIBRARY_PATH "/usr/X11R6/LessTif/Motif2.0/lib"

hole_server –p 1313 –e 10 –r 20 –t /home/disim/FTPOLK/models/terrain.smf925b.flt
-b ftpolk.dat –l 488360.0 3440792.0 15
```

This command executes the DTServer with the following settings:

-p 1313 : The DIS UPD port to communicate on

-e 10 : The DIS Exercise ID is set to 10

-r 20 : The rubble heartbeat is set to 20

-t /home/disim/FTPOLK/models/terrain.smf925b.flt : The ground terrain database file

-b ftpolk.dat : The terrain objects are read from the file ftpolk.dat

-l 488360.0 3440792.0 15 : The terrain map lower left easting and northing offsets and gridzone

6. Results

The DTServer was used in the Culminating Event for the Virtual Environments for Dismounted Soldier Simulation, Training, and Mission Rehearsal Science and Technology Objective at Ft. Benning, GA, in September 2002. The DTServer was used with the Fort Polk/Shugartt-Gordon database to provide dings and breaches.

The DTServer for this exercise was required to breach walls with the C4 explosive. One problem with the C4 was the lack of a velocity vector. The breaching algorithm used the velocity vector to compute the shape of the hole. The lack of a velocity vector caused the algorithm to abort. This was fixed by adding a velocity vector normal to the wall. Another issue for the Culminating Event was data dropouts. Due to the unreliable user datagram protocol message-passing protocol of the DIS standard, these dropouts caused the DTServer to not get a detonation, therefore not computing a breach. In this instance, the soldiers in the evaluation had to redo the C4 charge emplacement. Another case was more serious. This involved the situation where some soldier simulators received the Breach Subtract Set Data PDU and others did not. In this case, some simulators displayed the breach properly and some did not. This resulted in inconsistent databases on the network, which negatively impacted the training exercise. Using a reliable network protocol such as the High-Level Architecture would fix these problems.

Most simulation runs were successful. The soldiers in the training exercise used the ding mechanism as tracers and markers. Sniper location was marked by shooting at the wall outside the hide position, and breaching was utilized by the soldiers in accordance with doctrine for room clearing.

Another issue for the DTServer concerned the terrain database. The DTServer processes objects in the database for which it is programmed, namely buildings and the ground. Entities are ignored in the DTServer at this time. The scenarios in the Culminating Event contained entities for furniture and other building objects controlled by the semiautomated forces software, DISAF.

In one instance, the soldiers breached a wall only to find a Coca-Cola* machine in the way. In the real world, blast overpressure may have knocked the Coca-Cola machine down, or a large munition may have blown it up. These secondary effects need to be included in any future system for MOUT (military operations in urban terrain) simulations.

7. Conclusion

The DTServer provides real-time damage effects to distributed simulations. The DTServer has been interfaced to the Soldier Visualization System by "Reality By Design," the after-action review system by the Institute for Simulation and Training, and the ARL DIS. The DTServer provides logging and playback functionality for after-action review or predamaging a database from a previous run.

Future work will include upgrading to the High-Level Architecture, providing updates to late-arriving clients, and additional real-time munitions effects such as terrain cratering and effects on entities.

The mobility, line-of-sight, and obstacle creation provided by the DTServer can be applied to all high-resolution ground level entities such as robots, areal sensors, and unmanned ground vehicles.

* Coca-Cola is a registered trademark of the Coca-Cola Company.

Bibliography

Neiderer, A. M.; Thomas, M. A.; Pearson, R. *A Fracturing of Polygonal Object*; ARL-TR-1649, U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, April 1998.

Neiderer, A. M.; Hanson, C. E. *Distribution of Fragments Resulting From Polygonal Object Fracture*; ARL-TN-182, U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, September 2001.

Institute for Electrical and Electronic Engineers. *Standard for Distributed Interactive Simulation – Application Protocols*; DIS-4 Version 2.0, 4th draft (superseded by IEEE 1278.1); Institute for Simulation and Training: Orlando, FL, 4 February 1994.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
April 2003	Final	30 September 2001–30 September 2002

4. TITLE AND SUBTITLE
The U.S. Army Research Laboratory Dynamic Terrain Server

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Mark A. Thomas

5d. PROJECT NUMBER
P622783.Y103TEDNC

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-CT
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2962

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The U.S. Army Research Laboratory is conducting research in methods to compute and distribute dynamic terrain information in real-time distributed simulation. The purpose is to populate virtual environments with real-world dynamic terrain and objects such as rubble and debris, dings, and breaches of structures. This project is part of a science and technology objective to develop simulation capabilities for training dismounted soldiers using virtual simulation. This research resulted in the development of a dynamic terrain server, DTServer. DTServer computes results of battlefield munitions impacts on structures then transmits the results to receiving other simulations on the network using distributed interactive simulation protocol data units. Results of DTServer use in a dismounted infantry exercise at Ft. Benning, GA, demonstrate the utility of the tool. This report will describe the DTServer software and results of the dismounted infantry exercise.

15. SUBJECT TERMS
dynamic terrain, modeling and simulation, MOUT, distributed interactive simulation, High-Level Architecture

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Mark Thomas
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	22	19b. TELEPHONE NUMBER (Include area code) (410) 278-5011

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18